The ITOS Events API

Integrated Test & Operations System
5 October 2006

Introduction

ITOS includes an events subsystem which is responsible for collecting event messages from all processes in ITOS, logging them, displaying them, and distributing them. Applications within ITOS employ a simple API to send messages to the events subsystem, and this document describes that API and its operation.

The API is contained in the main ITOS library, '`libtcw.so`'. (A library containing only the Event API, called '`libitosevt.so`', also is available .)

1 Opening the Event Queue

To open a connection the the ITOS events subsystem, programs call the function `open_event()`. This function should be called before sending messages to the events subsystem.

This function opens a FIFO (also known as a *named pipe*) given by the environment variable `ITOS_EVTFIFO`. If `ITOS_EVTFIFO` is not set, the function tries to open '`./fifos/event_fifo`'. Upon opening the FIFO, it closes standard error, duplicates the FIFO file descriptor making the FIFO standard error, and closes the original FIFO file descriptor.

The `open_event()` function returns a 0 upon successful completion. On errors, it returns -1 and sets the global `char *liberrmsg` to a string indicating the error that occurred.

Note that if `open_event()` fails, applications can still call `EvtMsg()`, but the messages will go to the standard error output.

2 Sending Event Messages

To send event messages, ITOS programs call the function `EvtMsg()`, which has the following signature:

```
void EvtMsg(enum event_types event_type, char *format, ...)
```

and is used like the standard C function `printf()`. The *format* is a `printf()`-style format, with the extensions available to the ITOS function `Snprintf()`. The *event_type* is an enumerated constant giving the type of event being generated. The ITOS event windows allow user to filter messages based on these event types.

Details on the format extensions and event types are provided in the following sections.

2.1 How to use EvtMsg() and EvtMsgPurge()

The `EvtMsg()` function queues text within the calling application for output to the ITOS event FIFO. Text is actually written to the event FIFO when a newline character is queued, or when the *event_type* changes. In the latter case, the string '`[\n??]`' is appended at the end of the event message.

Some examples will help explain this behavior. The statements

```
EvtMsg(CMD_MSG,"This is a CMD_MSG event\n");
```

and

```
EvtMsg(CMD_MSG,"This is a ");
EvtMsg(CMD_MSG,"CMD_MSG event\n");
```

produce identical event messages. The statement

```
EvtMsg(IN_LIMITS,"One\nTwo\n");
```

produces two IN_LIMITS event messages. The statements

```
EvtMsg(RED_VIOL,"This is a ");
EvtMsg(YEL_VIOL,"YELLOW violation\n");
```

produces two event messages, the RED_VIOL message '`This is a [\n??]`' and the YEL_VIOL message '`YELLOW violation`'. Finally,

```
EvtMsg(TM_MSG,"%d + %d = 0x%08x\n", 15, 1, 16);
```

produces the TM_MSG message '`15 + 1 = 0x00000010`'.

An additional function, `EvtMsgPurge()`, clears any event message text queued in the local application. This might be used in cases where it is convenient to create the beginning of an error message just in case an error occurs. EvtMsgPurge() provides a way to discard that message if no error actually occurs.

2.2 What EvtMsg() writes

When `EvtMsg()` writes a message, it writes a two-digit ASCII integer corresponding to the *event_type* followed immediately, without space, by exactly what `printf()` (or, really, `Snprintf()`) would write.

From the examples in the previous section, then, the statement

```
EvtMsg(CMD_MSG,"This is a CMD_MSG event\n");
```

writes

```
11This is a CMD_MSG event\n
```

The '11' at the head of the output string is the event code. The numbers corresponding to the `enum event_types` constants can be found in Section 2.4 [Event Types], page 4.

2.3 Format Extensions

The following format extensions are available to users of `EvtMsg()`:

`%m` is replaced with the error message corresponding to the current value of the global error code `errno`, as it would be returned from the function `strerror()`. There must be no argument in the argument list corresponding to this conversion.

`%b` is used to print integer or unsigned values in binary representation.

2.4 Event Types

The *event_type* argument to `EvtMsg()` must be an enumerated constant from the following list. The interpretation of each event type is left to the programmer.

enum event_type	code	meaning
NULL_EVENT	00	Unknown event type; can't be filtered.
RED_VIOL	01	A red limits violation occurred.
YEL_VIOL	02	A Yellow limits violation occurred.
DEL_VIOL	03	A Delta limits violation occurred.
IN_LIMITS	04	A value went back in limits.
TM_MSG	05	Telemetry informational message.
TM_WARN	06	Telemetry warning message.
TM_ERROR	07	General telemetry error message.
CMD_EVENT	08	Command event.
CMD_VERIFY	09	Command verify/no-verify message.
CFG_ERROR	10	Configuration error message.
CMD_MSG	11	Command informational message.
CMD_WARN	12	Command warning message.
CMD_ERROR	13	General command error message.
CMD_TF	14	Command transfer frame echoed in hex.
OPER_ERROR	15	STOL Operator error.
STOL_ECHO	16	STOL echo of directives.
STOL_MSG	17	STOL MSG directive message.
STOL_WARN	18	STOL warning message.
STOL_ERROR	19	STOL error message.
DSP_MSG	20	Display informational message.
DSP_WARN	21	Display warning message.
DSP_ERROR	22	Display error message.
STOL_PRE	23	STOL directive preview message.

CMD_DECODE	24	Command decoded back from the spacecraft.
unused	25	unused.
SYS_ERROR	26	System call failure message – call a programmer!.
TCW_FAULT	27	Serious error message – call a programmer!.
SC_EVENT	28	Spacecraft event message.
CFG_ALERT	29	Configuration monitor alert message.
DEBUG_EVT	30	A debugging message.
SDP_MSG	31	Science Data Processing message.
SDP_WARN	32	Science Data Processing warning.
SDP_ERROR	33	Science Data Processing error.
CTLR_MSG	34	Controller message.
CTLR_WARN	35	Controller warning.
CTLR_ERROR	36	Controller error.
CFDP_MSG	37	CFDP Driver message.
CFDP_WARN	38	CFDP Driver warning.
FDP_ERROR	39	CFDP Driver error.

Index

Table of Contents